FYIT_24_Himanshu

IT-Tools ASSIGNMENT

PEP-8

1) Introduction ⇒ This document give coding convention for the python code comprising the standard library in the main python distribution. Please see the companion information PEP describing style guideline for the C code in the C implementation of python.
This document and PEP 257 (Docstring conventions) were adapted from guido original python Style Guide essay, with some addition from Barry's style guide.
This style Guide evolves over time as additional conventions are identified and post convention are rendered obsolete by change in the language itself.
many project have their coding style guideline.
In the event of any conflicts, such project specific Guide take precedence for that project.

2) A Foolish consistency is the Hobgoblin of Little minds ⇒

one of guido's key insight is that code is read much more often than it is written. The guideline provided here are intended to improve the ~~readabi~~ readability of code make it consistent across the wide Spectrum of python code As PEP 20 Say, "Readability Count.
A style guide is about consistency. consistency with this style guide is important. consistency within a project is more important Consistency withine one modular or function is the most important.
However, known when to be inconsistent - Some time style guide recommendation just aren't application. when in doubt use your best Judgement look at other example and decide what look best. And don't hesitate to ask!
in particular: do not break backward compatibility Just to comly with this PEP!
Some other good reason to ignore a particular ~~Just~~ guideline:

i) when applying the guideline would make the code less readable, even for Someone who is used to reading code that follows this PEP.

ii) To be consistent with Sourrouding code that also break it (may be for historicreason) although this is also an opportunity to clean up Someone else mess (in true XP Style).

iii) Because the code in question predates the introduction of the guideline and there is no other reason to be modifying that code.

iv) When the code needs to remain compatible with older version of python that don't Support the feature recommended by the Style guide.

3) Code Lay-Out ⇒ Indentation :- use 4 Space per indentation level.
Continuation line Should align wrapped element either vertically using python implicit line Joining inside parentheses, bracket and brace, or using a hanging indent [I]. When using a hanging indent the following Should be considered there Should be no argument on the first line and further indentation Should be used to clearly distinguish itself as a continuation line:

correct:

Aligned with opening delimiter.
foo =
long_function_name(var_one, var_two,
var_three, var_four)
Add 4 space (an extra level of indentation)
to distinguish argument from the rest
def long_function_name(var_one, var_two,
var_three, var_four):
print(var_one)
Hanging indents should add a level.
foo = long_function_name(var_one, var_two,
var_three, var_four)

Wrong:
Argument on first line forbidden
when not using vertical alignment
foo = long_function_name(var_one, var_two
var_three, var_four)
further indentation required as
indentation is not distinguishable.
def long_function_name(var_one, var_two, var_three
var_four):
print

The 4-space rule is optional for continuation
line

Optional:

# Hanging indent *may* be indented to other than 4 Space
foo = long_function_name(varone, var_two,
var three, var four)

when the conditional part of an if-statement is long enough to require that it be written across multiple line, its worth noting that the combination of a two character keyword plus a single space, plus an opening parenthesis create a natural 4-space indent for the subsequent line of the multiline condition. This can produce a visual conflict with the indented suite of code nested inside the if-statement which would also naturally be indented to 4 spaces. This PEP takes no explicit position on how to further visually distinguish such condition line from the nested suite inside the if-statement. Acceptable option in this situation include but are not limited to:

```
# No extra indentation
if (this_is_one_thing and that_is_another_thing):
    do_something()
# Add a comment, which will provide some
distinction in editors
# Supporting syntax highlight.
if (this_is_one_thing and that_is_another_thing):
# Since both condition are true. we can
fabricate do_something()
```

# Some extra indentation on the conditional continuation line
if (this_is_one_thing and that_is_another_thing): do_something()

(Also see the discussion of whether to break before or after binary operators below).

The closing brace/bracket/parenthesis on multiline constructs may either line up under the first non-whitespace character of the last line of list, as in:

```
my_list = [
    1, 2, 3,
    4, 5, 6
    ]

result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'b', 'e', 'f'
    )
```

or it may be lined up under the first character of the line that starts the multiline construct, as in:

```
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]

result = some_function_that_takes_arguments(
    'a', 'b', 'c', 'd', 'e', 'f',
    )
```

4) Tabs or Space ? ⇒ Space are the preferred idantion method.

Tabs Should be used Solely to remain Consistent with code that is already indented with tabs.
python disallow mixing tab and Space for indentation

5 maximum Line length ⇒ Limit all line to a maximum of 79 character
For flowing long blocks of text with fewer structural restriction (docstring or comment) the line length should be limited to 27 character.
limited to 72 characters.
limiting the required editor window width make possible to have several file open side by side, and work well when code review tool that present the two version in adjacent columns.
The default wraping in most tool disrupt the visual structure of the code, making it more difficult to understood. the limit are chosen to avoid wraping in editor with the window with set to 80, even if the tool place a marker glyph in the final column when wraping line. Some web based tool may not offer dyamic line wraping at all.

Some team strongley prefer a longer line length. for code maintained exclusily or primarily by a team that can reach agreement on this issue, it is okay to increase the line length limit upto 99 character provided that comment and docstring are still wrapped at 72 character.

The python standard library is conservative and require limit line to 79 character (and docstring/comment to 72)

The preferred way of wrapping long line is by using python limited implied line continuation inside parentheses, bracket and brace. Long line ~~is by using~~ python can be broken over multiple line by wraping expression in paranthese. these should be used in preference to using a backlash for line continuation.

Backslashes may still be appropriate at time for example, long, multiple with statement could not use implict continuation before python no 3.10, so backslashes were acceptable for that case.

```
with
open('/Path/to/some/file/you/want/to/read')
as file_1,
open('/Path/to/some/file/being/written', 'w')
as file_2:
file_2.write(file_1.read())
```

6) Should a line Break Before or After a Binary Operator ⇒

For decades the recommended style was to break after binary operation. But this can hurt readability in two way: the operator tend to get scattered across different column on the screen, and each operator is moved away from its operand and onto the previous line Here, the eye has to do extra work to tell which item are added and which are Subtracted.

# wrong:
# Operator Sit far away from their Operand
income = (gross_wages + taxable_interest + (dividends - qualified_dividend) - ira_deduction - student_loan_interest)

To Solve this readability problem, mathematician and their publisher follow the opposite convention Donald Knuth explain the traditional rule in His Computer and typesetting Series
"Although formulas within a para always break after binary operation and relation display formulas always break before binary operation

# Correct
# easy to match operator with operand
income = (gross_wages + taxable_interest + (dividends - qualified_dividend) - ira_deduction - student_loan_interest

in python code, it is permissible to break before or after a binary as long as the convention is consistent locally. For new code Knuth Style is suggested

7) Blank Line :→ Surround top-level function and class definition with two blank lines method definition inside a class are surround by a single blank line.
Extra blank line may be used (Sparingly) to separate group of related function Blank line may be omited between a bunch of related one-liner (e.g a set of dummy implementation).

use blank line function, sparingly, to indicate logical Selection

Python accept the control (i.e ^L) form feed character as whitespace many tool treat character as page separator. So you may use them to separate form feed will show an other glyph in its place

8) Source file Encoding ⇒ Code in the core python distribution should alway use UTF-8 and should not have an encoding declaration

in the standard libeirary, non-UTF-8 encoding should be used only for test purpose. use non-ASSCII character sparing preferably only to denote place and human name. if using non-ASCII character as data, avoid noisy unicode character like zalog and byte order mark

~~##~~

All identifier in the python standard libary must ASCII-only identifier, and should use English word wherever feasible (in many cases abbreviation and technical term are used which aren't english

Open source project with a global audience are encouraged to adopt a similar policy

9) import ⇒

import should usually be on Separate line:

```
# Correct:
import os
import sys

# wrong
import sys, os

its okay to say this though

# Correct:
from subprocess import Popen, PIPE
```

Import are always put at the top of the file, just after any module comment and docstring and before module global and constants.